STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

AFDELING MATHEMATISCHE STATISTIEK

S 392

Nul - Een Lineaire Programmering

de additieve algorithme van Balas

Jac. M. Anthonisse

maart 1968

The Mathematical Centre at Amsterdam, founded the 11th of February, 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications, and is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.) and the Central Organization for Applied Scientific Research in the Netherlands (T.N.O.), by the Municipality of Amsterdam and by several industries.

Inleiding

Dit rapport bevat een beschrijving van de additieve algorithme voor nul-een lineaire programmering zoals die in [1] door Balas werd gepubliceerd. De in [2, 3, 4, 5, 6, 7, 8] voorgestelde verbeteringen en opmerkingen worden niet besproken.
Voor details betreffende de algorithme zij verwezen naar de in de laatste paragraaf gegeven beschrijving in de vorm van een ALGOL-60 procedure.

Probleemstelling

Elk nul-een lineair programmeringsprobleem:

$$\text{minimaliseer} \quad \sum_{j=1}^{n} c'_j x'_j \tag{1}$$

onder de voorwaarden

$$\sum_{j=1}^{n} a'_{ij} x'_j \leq b'_i \quad (i = 1, \ldots, m_1) \tag{2}$$

$$\sum_{j=1}^{n} a'_{ij} x'_j \geq b'_i \quad (i = m_1 + 1, \ldots, m_2) \tag{3}$$

$$\sum_{j=1}^{n} a'_{ij} x'_j = b'_i \quad (i = m_2 + 1, \ldots, m') \tag{4}$$

$$x'_j = 0 \text{ of } 1 \quad (j = 1, \ldots, n) \tag{5}$$

is equivalent met:

$$\text{minimaliseer} \quad z = \sum_{j=1}^{n} c_j x_j \tag{6}$$

onder de voorwaarden

$$\sum_{j=1}^{n} a_{ij} x_j + y_i = b_i \quad (i = 1, \ldots, m) \tag{7}$$

$$x_j = 0 \text{ of } 1 \quad (j = 1, \ldots, n) \tag{8}$$

$$y_i \geq 0 \quad (i = 1, \ldots, m) \tag{9}$$

waarbij $c_j \geq 0 \quad (j = 1, \ldots, n).$ (10)

Overgang van (1) t/m (5) op (6) t/m (10) wordt bereikt door:

a) elk van de voorwaarden uit (4) te vervangen door twee voorwaarden en wel

$$\left.\begin{aligned} \sum_{j=1}^{n} a'_{ij} x'_j &\leq b'_i \\ \sum_{j=1}^{n} a'_{ij} x'_j &\geq b'_i \end{aligned}\right\} \quad (i = m_2 + 1, \ldots, m'),$$

zodat nu alle voorwaarden van type (2) of (3) zijn.

b) vermenigvuldiging van alle voorwaarden van type (3) met -1, zodat deze overgaan in voorwaarden van type (2).

c) te substitueren

$$x'_j = \begin{cases} x_j & \text{als } c'_j \geq 0 \\ 1 - x_j & \text{als } c'_j < 0 \end{cases} \qquad (j = 1, \ldots, n)$$

waardoor $c'_j x'_j$ met $c'_j < 0$ overgaat in

$$c'_j(1 - x_j) = c'_j + (-c'_j)x_j.$$

De termen c'_j doen voor het minimalisatie-proces niet terzake en zijn daarom in (6) weggelaten.

Op grond van de volgende overwegingen kan het aantal variabelen x_j soms worden gereduceerd:

Is $x_j = \bar{x}_j$ een oplossing van (6) t/m (10) dan is ook

$x_j = \bar{x}'_j$ een oplossing van (6) t/m (10), waarbij

$$\bar{x}'_j = \begin{cases} 0 & \text{als } a_{ij} \geq 0 \quad (i = 1, \ldots, m) \\ \bar{x}_j & \text{anders} \end{cases} \qquad (j = 1, \ldots, n) \quad (11)$$

Stel $\bar{x}_{j*} = 1$ en $a_{ij*} \geq 0$ $(i = 1, \ldots, m)$. Door toepassing van (11) voor $j = j^*$ gaat

$$z = \sum c_j \bar{x}_j \quad \text{over in} \quad z - c_{j*} \quad \text{en gaat}$$

$$y_i = b_i - \sum a_{ij} \bar{x}_j \quad \text{over in} \quad y_i + a_{ij*} \quad (i = 1, \ldots, n).$$

zodat de waarde van z niet stijgt en geen enkele y_i negatief wordt. Dit betekent dat de x_j waarvoor geldt

$$a_{ij} \geq 0 \quad (i = 1, \ldots, m) \qquad (12)$$

mogen worden geschrapt.

In het nu volgende wordt verondersteld dat een bovengrens z^* (bijv. $z^* = +\infty$) voor de minimale waarde van z bekend is, zodat gezocht wordt naar oplossingen van (7) t/m (10) met $z < z^*$, zodra een oplossing is gevonden wordt z^* gelijk gesteld aan de z-waarde van die oplossing, en het zoeken (nu met de nieuwe z^*) wordt voortgezet.

Globale omschrijving van de algorithme

Elke verzameling indices $J_q = \{j_1, \ldots, j_q\}$ met

$$j_l \in \{1, 2, \ldots, n\} \quad (l = 1, \ldots, q)$$

bepaalt een oplossing van (7) en (8) door te stellen:

$$\begin{cases} x_j = 1 & \text{als } j \in J_q \\ x_j = 0 & \text{als } j \notin J_q \end{cases}$$

zodat

$$y_i = b_i - \sum_{l=1}^{q} a_{ij_l} \qquad (i = 1, \ldots, m)$$

en

$$z = \sum_{l=1}^{q} c_{j_l}.$$

Voor q = 0 geeft dit een triviale oplossing van (7) en (8):

$$x_j = 0 \qquad (j = 1, \ldots, n)$$

$$y_i = b_i \qquad (i = 1, \ldots, m)$$

$$z = 0.$$

Indien $b_i \geq 0$ $(i = 1, \ldots, m)$ in deze triviale oplossing een optimale oplossing voor (6) t/m (10), omdat uit (10) volgt dat

$$\sum_{j=1}^{n} c_j x_j \geq 0 \text{ voor alle oplossingen van (8).}$$

De algorithme bestaat uit het genereren van geneste indexverzamelingen

$$J_0 \subset J_1 \subset \ldots \subset J_q \subset \ldots$$

tot A) of B) optreedt:

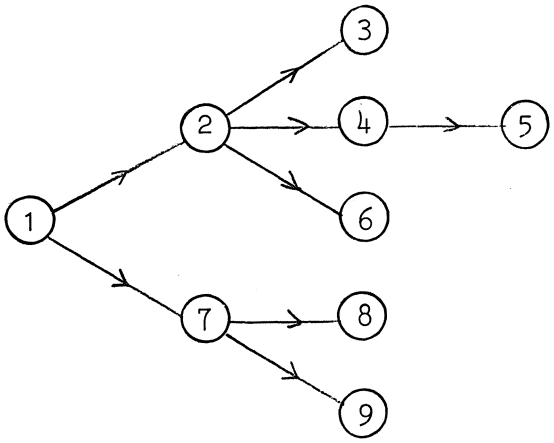
A) een J_q is bereikt die voldoet aan (9) en waarvoor $z < z^*$,

B) een J_q is bereikt die niet aan (9) voldoet maar waarvan het zeker is dat iedere uitbreiding óf nooit tot een index-verzameling zal leiden die wel aan (9) voldoet óf slechts tot oplossingen met $z \geq z^*$,

waarna van het uitbreidende (forward) naar het inkrimpende (backward) deel van de algorithme wordt overgegaan.

Dit inkrimpende deel van de algorithme bestaat uit het ongedaan maken van de laatste uitbreiding en het zoeken van een andere uitbreiding. Treedt voor een gevonden andere uitbreiding A) of B) op dan wordt ook deze uitbreiding weer ongedaan gemaakt en vervangen door een die van de voorgaande uitbreidingen verschilt.
Zijn voor de laatste uitbreiding geen vervangers meer beschikbaar, dan wordt ook de voorlaatste uitbreiding ongedaan gemaakt en vervangen etc.

De gang van zaken wordt in de volgende figuren weergegeven. Elk cirkeltje stelt een indexverzameling voor, (K)→(L) betekent dat indexverzameling L een uitbreiding is van indexverzameling K. De verzamelingen worden in volgorde van nummering afgewerkt.



Uit deze omschrijving van de algorithme blijkt direct dat een optimale oplossing van (6) t/m (10), indien deze bestaat, in een eindig aantal stappen wordt gevonden.
Immers:
er zijn juist 2^n indexverzamelingen, die alle expliciet (als een J_q) of impliciet (als niet gegenereerde uitbreiding van een J_q) worden onderzocht. Wegens A) en B) kan onder de niet gegenereerde uitbreidingen van een J_q geen of geen betere oplossing van (7) t/m (10) worden gevonden.

De uitbreidingen

Zij $J_q = \{j_1, \ldots, j_q\}$ een gegenereerde indexverzameling. Door J_q is een oplossing van (7) en (8) gedefinieerd met

$$y_i = b_i - \sum_{l=1}^{q} a_{ij_l} \qquad (i = 1, \ldots, m)$$

en

$$z = \sum_{l=1}^{q} c_{j_l}.$$

Als $y_i \geq 0 \qquad (i = 1, \ldots, m)$

treedt situatie A) op, z^* krijgt een nieuwe waarde en de algorithme wordt voortgezet bij het inkrimpende gedeelte.
Voor het vervolg van deze paragraaf wordt aangenomen dat $y_i < 0$ voor tenminste één index i.

Uitbreiding van J_q met j_{q+1} tot

$$J_{q+1} = \{j_1, \ldots, j_q, j_{q+1}\}$$

geeft een indexverzameling waarvoor geldt:

$$z' = z + c_{j_{q+1}}$$

en

$$y'_i = y_i - a_{ij_{q+1}} \qquad (i = 1, \ldots, m).$$

Hieruit volgt direct dat een index j met:

$$c_j \geq z^* - z \qquad (13)$$

niet in aanmerking komt om aan J_q te worden toegevoegd. Indices j die aan (13) voldoen komen ook niet in aanmerking om aan een uitbreiding van J_q te worden toegevoegd.
Verder geldt:

$$y'_i > y_i \text{ dan en slechts dan als } a_{ij_{q+1}} < 0,$$

zodat een index j met

$$a_{ij} \geq 0 \quad \text{voor alle } i \text{ met } y_i < 0$$

niet in aanmerking komt om aan J_q te worden toegevoegd.
Vanzelfsprekend mag een index $j \in J_q$ niet nogmaals aan J_q worden toegevoegd.
Zij I de verzameling indices die niet tot J_q behoren en niet op grond van (13) zijn uitgesloten.
Definieer:

$$d_i = \sum_j (a_{ij} | a_{ij} < 0,\ j \in I) \quad (i = 1, \ldots, m),$$

$-d_i$ kan worden geinterpreteerd als de maximale 'groei' van y_i die mogelijk is door J_q uit te breiden.

Beschouw de voorwaarden met $y_i < 0$, d.w.z. de voorwaarden waaraan door J_q niet is voldaan.

De volgende situaties kunnen zich voordoen:

α) er is een i waarvoor geldt: $y_i < 0$, $y_i < d_i$, dus $y_i - d_i < 0$.
Dit betekent dat geen enkele uitbreiding van J_q een oplossing geeft die aan (9) voldoet (situatie B).

β) er is een i met: $y_i < 0$, $y_i = d_i$, zodat $y_i - d_i = 0$.
In dit geval bevat elke uitbreiding van J_q die aan (9) voldoet alle indices j met $a_{ij} < 0$, $j \in I$, J_q moet dus met deze indices worden uitgebreid, tenzij hierdoor $z \geq z^*$ zou worden (situatie B).

γ) voor alle i met $y_i < 0$ is $y_i > d_i$, dus $y_i - d_i > 0$.
Vanuit deze situatie worden de indices $j \in I$ nader onderzocht.

Zij $j_1 \in I$, en i de index van een voorwaarde met

$$0 \leq y_i < a_{ij_1}.$$

Door toepassing van j_1 aan J_q gaat y_i over in

$$y'_i = y_i - a_{ij_1} < 0,$$

met bijbehorende

$$d'_i = \sum_j (a_{ij} | a_{ij} < 0,\ j \in I'),$$

waarbij I' uit de verzameling indices j bestaat die niet tot J_{q+1} = $= J_q \cup \{j_1\}$ behoren en waarvoor $c_j < z^* - z'$.
Wegens $z' \geq z$ en $a_{ij_1} > 0$ is dan

$$d'_i \geq d_i.$$

Veronderstel dat geldt:

$$y_i < d_i + a_{ij_1}$$

dan is

$$y'_i = y_i - a_{ij_1} < d_i \leq d'_i$$

dus

$$y'_i < d'_i.$$

hieruit volgt:
Een index $j \in I$ waarvoor een index i bestaat met

$$0 \leq y_i < d_i + a_{ij} \qquad (14)$$

komt niet in aanmerking om aan J_q te worden toegevoegd. De indices j die aan (14) voldoen komen ook niet in aanmerking om aan een uitbreiding van J_q te worden toegevoegd.
Uit de verzameling indices die na alle bovengenoemde tests nog in aanmerking komen om aan J_q te worden toegevoegd wordt een index j waarvoor

$$v_j = \sum_i (y_i - a_{ij} | y_i \leq a_{ij})$$

maximaal is gekozen en aan J_q toegevoegd. Op grond van de J_{q+1} die ontstaat worden bovenstaande berekeningen herhaald.

De inkrimpingen

Zoals reeds hierboven beschreven bestaat de algorithme uit het genereren van geneste indexverzamelingen.

$$J_0 = \emptyset \subset J_1 \subset \ldots \subset J_q \subset J_{q+1} \subset \ldots .$$

Vanuit J_0 wordt in maximaal n uitbreidingen een indexverzameling gevonden die niet verder behoeft te worden uitgebreid. Zij J_{q+1} deze indexverzameling, veronderstel dat J_{q+1} uit J_q is ontstaan door toevoeging van de index j_{q+1}.

De inkrimping van J_{q+1} bestaat uit het ongedaan maken van de uitbreiding waardoor J_{q+1} is ontstaan, dus J_q wordt geregenereerd, zodat

$$z' = z - c_{j_{q+1}}$$

en

$$y'_i = y_i + a_{ij_{q+1}} \qquad (i = 1, \ldots, m).$$

Nu wordt het uitbreidende gedeelte van de algorithme weer toegepast om een J'_{q+1} te bepalen. Om 'cycling' te voorkomen moet hierbij worden verzekerd dat de index j_{q+1} in geen van de nieuwe uitbreidingen van J_q en J'_{q+1} voorkomt.

Dit kan als volgt worden georganiseerd:

Zij

$$w_j^q = \begin{cases} v_j & \text{als index } j \text{ in aanmerking komt om aan } J_q \text{ te worden toegevoegd,} \\ +2 & \text{als index } j \text{ noch aan } J_q, \text{ noch aan een uitbreiding van } J_q \text{ mag worden toegevoegd,} \\ +1 & \text{anders.} \end{cases}$$

Wegens $v_j \leq 0$ zijn de indices hiermee ondubbelzinnig onderscheiden.

Zodra j aan J_q wordt toegevoegd wordt de waarde (v_j) van w_j^q vervangen door +2. (15)

Bij het bepalen van de indices die in aanmerking komen om aan J_{q+1} te worden toegevoegd kunnen de j met $w_j^q = 2$ zonder meer worden uitgesloten,

hetgeen de berekeningen in het uitbreidende deel van de algorithme zeer kan bekorten.

Is de toevoeging van j aan J_q ongedaan gemaakt dan wordt als volgende uitbreiding een index j gekozen waarvoor

$$(w_j^q | w_j^q \leq 0)$$

maximaal is en

$$c_j < z^* - z.$$

Als geen uitbreiding van J_q meer mogelijk is, en $q > 0$, dan wordt J_{q-1} geregeneerd en opnieuw uitgebreid. Indien $q = 0$ is de algorithme beëindigd.

Ter verduidelijking van deze organisatie de volgende figuren:

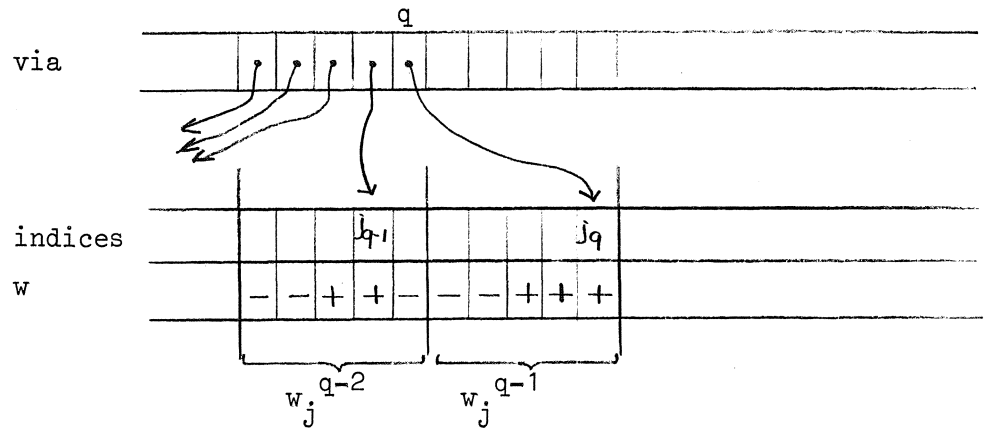


fig. 1.

De q-de plaats van de lijst 'via' heeft als waarde een plaatsnummer van de lijsten 'indices' en 'w'. De aangewezen plaats in 'indices' heeft als waarde j_q, de overeenkomstige plaats in 'w' heeft als waarde w_j^{q-1}.
Bovenstaande figuur geeft de situatie weer onmiddellijk na de toevoeging van j_q aan J_{q-1}. De waarde van de plaatsen in 'via' met een hoger nummer dan q (rechts van de q-de plaats), evenals die van de plaatsen rechts van de w_j^{q-1} is niet gedefinieerd.

Na berekening van de w_j^q is de situatie als volgt:

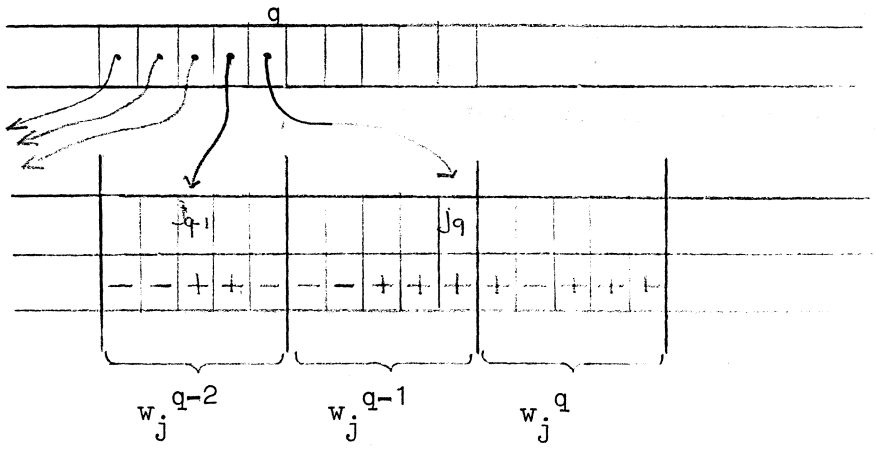


fig. 2.

Op grond van de w_j^q wordt nu j_{q-1} gekozen en aan J_q toegevoegd:

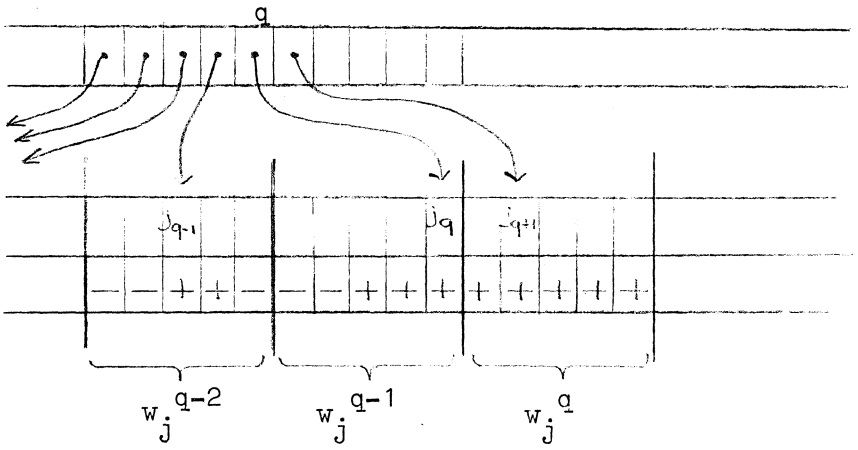


fig.3.

Stel nu dat uitbreiding van J_{q+1} onnodig (situatie A) of onmogelijk (situatie B) blijkt te zijn, dus de toevoeging van j_{q+1} aan J_q wordt ongedaan gemaakt:

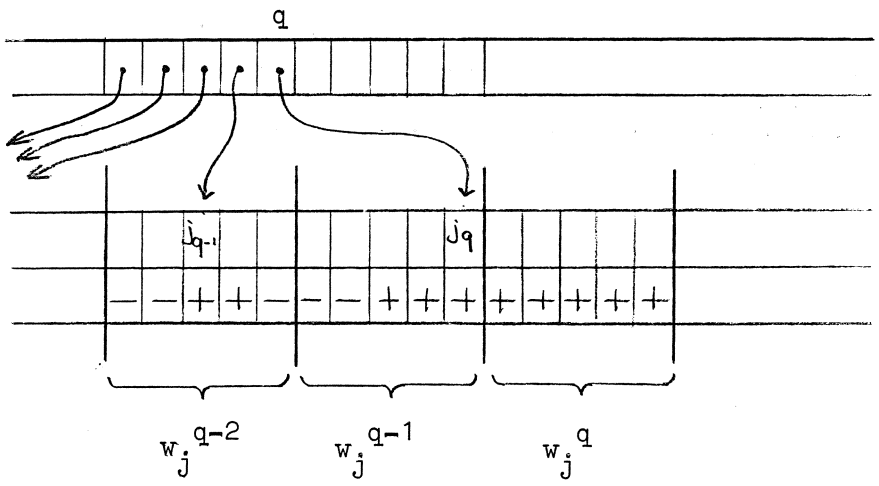


fig. 4.

Daar $w_j^q > 0$ voor elke j is een nieuwe uitbreiding van J_q niet mogelijk, dus wordt ook de toevoeging van j_q aan J_{q-1} ongedaan gemaakt, waarbij de w_j^q kunnen worden vergeten:

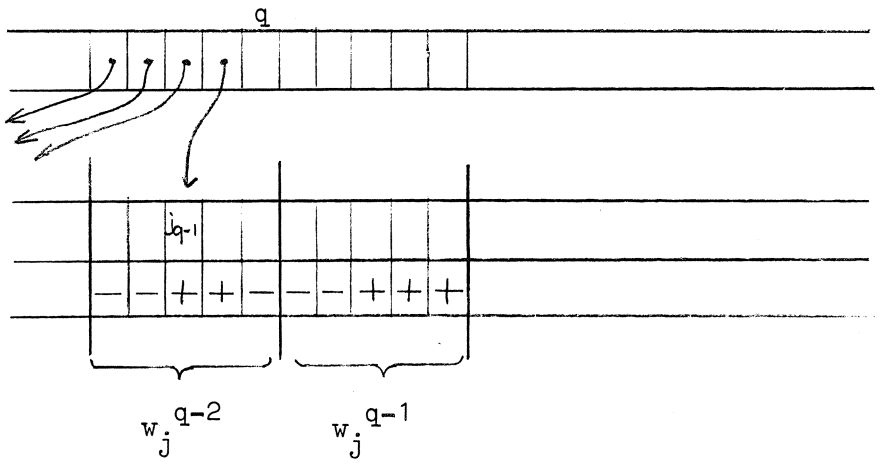


fig. 5.

Een nieuwe uitbreiding van J_{q-1} blijkt mogelijk te zijn:

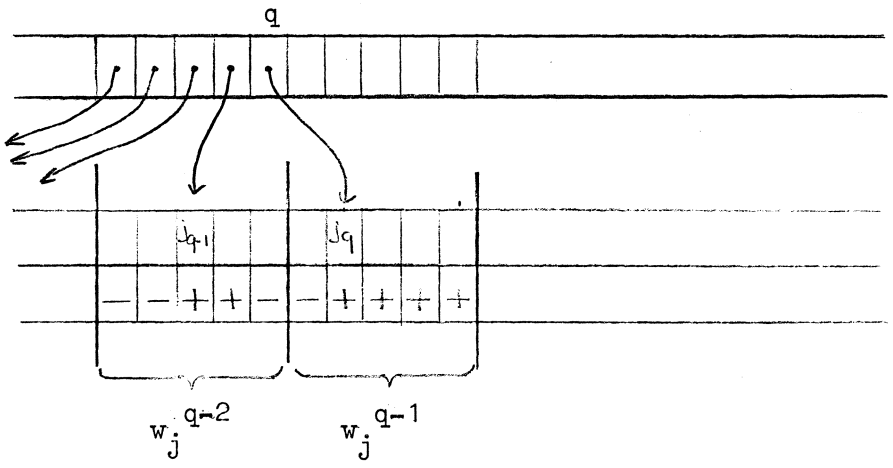


fig. 6.

Uit vergelijking van fig. 6 met fig. 1 blijkt dat beide situaties geheel analoog zijn. Nu kunnen dus weer getallen w_j^q worden berekend, die zullen verschillen van de w_j^q in fig. 2 daar zij zijn gebaseerd op een nieuwe indexverzameling J_q.

Literatuur

1. Egon Balas, An Additive Algorithm for Solving Linear Programs with Zero-One Variables.
Operations Research, 13(1965)4, 517-546.

2. Fred Glover and Stanley Zionts, A note on the Additive Algorithm of Balas.
Operations Research, 13(1965)4, 546-549.

3. Raoul J. Freeman, Computational Experience with a 'Balasian' Integer Programming Algorithm.
Operations Research 14(1966)5, 935-940.

4. Willard D. James, Some Remarks on a Numerical Example of Balas and the Note by Glover and Zionts.
Operations Research, 14(1966)5, 941-941.

5. Egon Balas, Comment on the Preceding Note.
Operations Research, 14(1966)5, 942-942.

6. B. Fleischman, Computational Experience with the algorithm of Balas.
Operations Research, 15(1967)1, 153-155.

7. C.C. Petersen, Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R & D projects.
Management Science, 13(1967)9, 736-750.

8. Karl M. Brauer, A Note on the Efficiency of Balas' Algorithm.
Operations Research, 15(1967)6, 1169-1171.

```
integer procedure BALASADDITIVE(A, B, C, X, n, m, z, infeasible, max);
value B, n, m, z, max; integer n, m, z, max; integer array A, B, C, X;
label infeasible;
comment BALASADDITIVE:= minimal value of CX under the conditions
that AX ≤ B, CX < z and the elements of X have the value zero or
one. If no such solution exists the call of the procedure results
in a jump to infeasible. Otherwise the solution is stored into the
array X[1:n]. The array C[1:n] should contain non-negative numbers.
No such restriction is imposed on the elements of A[1:m, 1:n] and
B[1:m]. max = a large number (10^6);

begin integer q, j, zz, s, i, i1, i2, d, a, h, y, k, v, w;
      Boolean solved, not;
      Boolean array F[1:n];
      integer array VIA, S[1:n], D[1:m], IND, W[1:n × (n + 1) : 2],
      impr[ - 1:n];
      q:= impr[0]:= 0; solved:= false; impr[ - 1]:= 0;
      for j:= 1 step 1 until n do F[j]:= true; zz:= 0; s:= 0;
iter: for i:= 1 step 1 until m do if B[i] < 0 then goto step0;
      goto sol;
step0: i1:= impr[q - 1] + 1; i2:= impr[q]; d:= z - zz;
      for h:= i1 step 1 until i2 do
      begin j:= IND[h]; a:= W[h]; F[j]:= a ≠ 2; end;
      for j:= 1 step 1 until n do if F[j] then F[j]:= C[j] < d;
      i1:= i2 + 1;
      for j:= 1 step 1 until n do if F[j] then
      begin i2:= i2 + 1; IND[i2]:= j; W[i2]:= 1 end;
      if i1 = i2 then goto back; q:= q + 1; impr[q]:= i2;
      for i:= 1 step 1 until m do D[i]:= 0;
      for h:= i1 step 1 until i2 do
      begin j:= IND[h];
            for i:= 1 step 1 until m do
            begin a:= A[i,j]; if a < 0 then D[i]:= D[i] + a end
      end;
      for i:= 1 step 1 until m do
```

```
        begin y:= B[i]; if y ≥ 0 then goto nexti; d:= D[i];
              if y < d then
              begin q:= q - 1; goto back end;
              if y = d then goto step2;
        nexti:
        end;
        k:= 0; w:= - max;
step1:  for h:= i1 step 1 until i2 do
        begin j:= IND[h]; v:= 0; not:= true;
              for i:= 1 step 1 until m do
              begin d:= D[i]; y:= B[i]; a:= A[i,j];
                    if 0 ≤ y ∧ y < d + a then
                    begin v:= 2; goto next end;
                    if y < 0 ∧ a < 0 then not:= false;
                    if y < a then v:= v + y - a
              end;
              if not then v:= 1;
        next: W[h]:= v; if v ≤ 0 ∧ v > w then
              begin k:= h; w:= v end;
         end;
in:     if k = 0 then goto back; s:= s + 1; S[s]:= q; VIA[s]:= k;
        j:= IND[k]; zz:= zz + C[j]; W[k]:= 2;
        for i:= 1 step 1 until m do B[i]:= B[i] - A[i,j]; goto iter;
step2:  for h:= i1 step 1 until i2 do
        begin j:= IND[h]; if A[i,j] ≥ 0 then goto no; s:= s + 1;
              S[s]:= q; VIA[s]:= h; j:= IND[k]; zz:= zz + C[j];
              W[h]:= 2;
              for i:= 1 step 1 until m do B[i]:= B[i] - A[i,j];
              if zz ≥ z then goto back;
        no:
        end;
        goto iter;
sol:    z:= zz;
        for j:= 1 step 1 until n do X[j]:= 0;
        for h:= 1 step 1 until s do X[IND[VIA[h]]]:= 1; solved:= true;
back:   if s = 0 then goto out; if VIA[s] < q then goto step3;
        j:= IND[VIA[s]]; zz:= zz - C[j];
```

```
        for i:= 1 step 1 until m do B[i]:= B[i] + A[i,j]; s:= s - 1;
        goto back;
step3:  k:= 0; w:= - max; i1:= impr[q - 1] + 1; i2:= impr[q];
        for h:= i1 step 1 until i2 do
        begin j:= IND[h]; a:= W[h]; if a > 0 then goto nc;
              if C[j] + zz ≥ z then
              begin W[h]:= 2; goto nc end;
              if a > w then
              begin k:= h; w:= a end;
        nc:
        end;
        if k ≠ 0 then goto in;
        for h:= i1 step 1 until i2 do F[IND[h]]:= true; q:= q - 1;
        goto back;
out:    if solved then BALASADDITIVE:= z else goto infeasible
end;
```

Correctie bij rapport S 392.

De in bovengenoemd rapport gegeven ALGOL-60 procedure BALASADDITIVE bevat enkele fouten.
Ondanks deze fouten werden de problemen waarmee de procedure is getest correct opgelost. Een gewijzigde en met andere problemen geteste procedure gaat hierbij.

2-4-1968
Jac. M. Anthonisse

```
integer procedure BALASADDITIVE(A, B, C, X, n, m, z, infeasible, max);
value B, n, m, z, max; integer n, m, z, max; integer array A, B, C, X;
label infeasible;
comment BALASADDITIVE:= minimal value of CX under the conditions
that AX ≤ B, CX < z and the elements of X have the value zero or
one. If no solution exists the call of the procedure results in a
jump to infeasible. Otherwise the solution is stored into the array
X[1:n]. The array C[1:n] should contain non-negative numbers. No
such restriction is imposed on the elements of A[1:m, 1:n] and
B[1:m]. max = a large number (10 6);

begin integer q, j, zz, s, i, i1, i2, d, a, h, y, k, v, w;
      Boolean solved, not;
      Boolean array F[1:n];
      integer array S[0:n], VIA[1:n], D[1:m], IND, W[1:n × (n + 1) :
      2], impr[ - 1:n];
      q:= impr[0]:= impr[ - 1]:= S[0]:= s:= zz:= 0; solved:= false;
      for j:= 1 step 1 until n do F[j]:= true;
iter: for i:= 1 step 1 until m do if B[i] < 0 then goto step0;
      goto sol;
step0: i1:= impr[q - 1] + 1; i2:= impr[q]; d:= z - zz;
      for h:= i1 step 1 until i2 do
      begin j:= IND[h]; a:= W[h]; F[j]:= a ≠ 2; end;
      for j:= 1 step 1 until n do if F[j] then F[j]:= C[j] < d;
      i1:= i2 + 1;
      for j:= 1 step 1 until n do if F[j] then
      begin i2:= i2 + 1; IND[i2]:= j; W[i2]:= 1 end;
      if i1 > i2 then goto back; q:= q + 1; impr[q]:= i2;
      for i:= 1 step 1 until m do D[i]:= 0;
      for h:= i1 step 1 until i2 do
      begin j:= IND[h];
            for i:= 1 step 1 until m do
            begin a:= A[i,j]; if a < 0 then D[i]:= D[i] + a end
      end;
      for i:= 1 step 1 until m do
```

```
        begin y:= B[i]; if y ≥ 0 then goto nexti; d:= D[i];
              if y < d then
              begin q:= q - 1; goto back end;
              if y = d then goto step2;
        nexti:
        end;
        k:= 0; w:= - max;
step1:  for h:= i1 step 1 until i2 do
        begin j:= IND[h]; v:= 0; not:= true;
              for i:= 1 step 1 until m do
              begin d:= D[i]; y:= B[i]; a:= A[i,j];
                    if 0 ≤ y ∧ y < d + a then
                    begin v:= 2; goto next end;
                    if y < 0 ∧ a < 0 then not:= false;
                    if y < a then v:= v + y - a
              end;
              if not then v:= 1;
        next: W[h]:= v; if v ≤ 0 ∧ v > w then
              begin k:= h; w:= v end;
         end;
in:     if k = 0 then goto back; s:= s + 1; S[s]:= q; VIA[s]:= k;
        j:= IND[k]; zz:= zz + C[j]; W[k]:= 2;
        for i:= 1 step 1 until m do B[i]:= B[i] - A[i,j]; goto iter;
step2:  for h:= i1 step 1 until i2 do
        begin j:= IND[h]; if A[i,j] ≥ 0 then goto no; s:= s + 1;
              S[s]:= q; VIA[s]:= h; zz:= zz + C[j]; W[h]:= 2;
              for k:= 1 step 1 until m do B[k]:= B[k] - A[k,j];
              if zz ≥ z then goto back;
        no:
        end;
        goto iter;
sol:    z:= zz;
        for j:= 1 step 1 until n do X[j]:= 0;
        for h:= 1 step 1 until s do X[IND[VIA[h]]]:= 1; solved:= true;
back:   if q = 0 then goto out; if S[s] < q then goto step3;
        j:= IND[VIA[s]]; zz:= zz - C[j];
        for i:= 1 step 1 until m do B[i]:= B[i] + A[i,j]; s:= s - 1;
```

```
        goto back;
step3:  k:= 0; w:= - max; i1:= impr[q - 1] + 1; i2:= impr[q];
        for h:= i1 step 1 until i2 do
        begin j:= IND[h]; a:= W[h]; if a > 0 then goto nc;
              if C[j] + zz ≥ z then
              begin W[h]:= 2; goto nc end;
              if a > w then
              begin k:= h; w:= a end;
        nc:
        end;
        if k ≠ 0 then goto in;
        for h:= i1 step 1 until i2 do F[IND[h]]:= true; q:= q - 1;
        goto back;
out:    if solved then BALASADDITIVE:= z else goto infeasible
end BALASADDITIVE;
```